DEIM Forum 2013 A9-1

類似度の階層関係に基づく木構造索引を用いた効率的な類似検索

劉　健全† 西村　祥治† 荒木　拓也†

† 日本電気株式会社クラウドシステム研究所 〒 211–8666 神奈川県川崎市中原区下沼部 1753
E-mail: †j-liu@ct.jp.nec.com, ††s-nishimura@bk.jp.nec.com, †††t-araki@dc.jp.nec.com

あらまし　本論文では，高次元の類似検索問題に対して，類似度の階層関係に基づく木構造索引と，これを用いた検索アルゴリズムを提案する．従来研究では，多くの索引手法がデータの空間性質と距離関数の幾何性質を利用して，空間索引を構築することにより，多次元類似検索の検索効率を向上させる．しかし，データの次元数が増えることに伴い，索引構造の内部にデータのオーバーラップが大量に発生するため，検索効率が急激に低下するという問題が発生する．本研究では，データの内部属性に依存せず，類似度関数または距離関数の内部計算に依存せずに，データ間の類似度の階層関係に基づいて，高次元にたえられる新たな木構造索引を提案する．また，提案する索引を用いた類似検索アルゴリズムを提案する．さらに，合成データと実データを用いて提案手法の有効性と検索の効率を検証する．

キーワード　類似度索引，空間索引，類似検索，最近傍検索，範囲問合せ

1. はじめに

過去十年間，類似検索の科学研究・開発への応用が急増している．例として，パターン認識や [1], [2]，画像検索 [3]，時系列マッチング [4] などが挙げられる．これらの応用に伴って，類似検索に関する研究は活発にされている．類似検索に関する問合せ処理は，k 近傍 (k-Nearest Neighbors, k-NN) や，範囲問合せ (Range Query)，逆最近傍 (Reverse Nearest Neighbors) などの変形がある．これらに対するオンライン問合せ処理の要求が急増しており，様々な処理技術が提案された．例えば，k-d-b-tree [5] や，R-tree [6] などの空間索引技術が挙げられる．しかしながら，これらの空間索引手法では，高次元データを対象として扱う場合，一般に「次元の呪い (the curse of dimensionality)」と呼ばれる高次元ならではの性質により，検索性能が低下するという問題が起こる．この性質のため，簡単な連続スキャン手法と比較して，空間分割やデータ分割などの索引技術ではむしろ効果が悪くなることが指摘されている [7].

一方，ビックデータの時代に，高次元データを扱う実応用は多くある．代表として，画像や，音声，映像のような実データを対象とする検索の需要は，近年急激に増えている．画像・音声・映像の検索は，元の画像・音声・映像から抽出された特徴量を用いて，特徴量間の類似性を算出したうえで，クエリ画像・音声・映像と類似したデータを探し出す．一般的に，このような特徴量は通常の数値型の低次元データと異なって，数百から数万次元までの高次元データになる．特徴量データは，一般的に数値ベクトルとして表現するが，商用性の高い製品では，計算上の性能を追求する，または，技術秘密を保護するため，特別な手法でシリアライズ化されたバイト列で表現することも少なくない．一例として，**NeoFace®**(注1) は顔検出または顔照合を行うアプリケーションを作成するための商用開発キットである．**NeoFace®** は顔照合の速さと精度で世界一の誇りを持つため(注2)(注3)，顔画像を用いた映像監視システムでは，映像フレームの類似検索に高精度の顔照合を行うために **NeoFace®** は導入される．しかし，**NeoFace®** で扱う顔の特徴量データは保護されたバイト列であり，顔間の類似照合を行うにも開発キットが提供した照合 API を介することになる．

しかし，前述した空間索引技術は，特徴量データにおける類似検索を高速に行うため，特徴量データの幾何空間性質，あるいは，類似度を計算する類似度関数または距離関数が満たせる三角不等式などの性質を利用して，空間索引の構築と索引における検索を行う．例えば，R-tree は索引を作成する際に，特徴量データに内部の各次元の数値座標を基に，最小包含矩形 (MBR) を計算する必要である．さらに，検索時には，MBR を利用して木の走査に対して，距離関数が満たせた三角不等式を基に索引ノードまたはデータの枝刈りを行う必要である．このために，既存の空間索引手法では，索引を構築するために，特徴量データに内部の各次元値を知る必要

(注1)：http://www.nec.co.jp/soft/neoface/product/neoface.html
(注2)：http://www.nist.gov/customcf/get_pdf.cfm?pub_id=905968
(注3)：http://www.nec.co.jp/press/ja/1006/3002.html

であり，類似度を計算するために，三角不等式の性質を満たせるかを判断するにも，類似度関数または距離関数の内部演算を知る必要である．よって，上述した **NeoFace®** のような開発キットを導入した類似検索システムを開発するには，特徴量データの内部構造および類似照合関数の内部演算が入手できないので，既存の空間索引技術が応用できない問題が生じる．

前述のような内部構造未明の高次元データを対象とする類似検索システムの実応用開発にあたって，本研究では，下記の類似検索問題に着眼している．

$$SS(q, \delta, DB) = \{o \mid o \in DB \ \wedge \ sim(q, o) \geqq \delta\}.$$

そこで，類似検索 SS は，与えられたクエリデータ q に対して，データベース DB の中から，既定の類似度を計算する関数 $sim(\cdot, \cdot)$ を用いて，指定した類似度閾値 δ 以上のデータ o の集合を探し出す．データの内部構造は従来の数値ベクトルを含め限定しないが，本論文では，既存技術が対応できないバイト列のような高次元データを中心に議論する．また，類似度計算関数 $sim(\cdot, \cdot)$ についても，従来研究で多く利用されるコサイン関数 ($\cos(\cdot, \cdot)$) などを含め限定しないが，本稿では，2 つのデータだけを与えて算出される類似度値を返すブラックボックスのような類似照合 API を中心に議論する．

従って，既存の空間索引技術は我々が着眼する類似検索問題に応用できないため，本研究では，新たなデータ索引手法（**類似度木**）を提案する．類似度木という索引は，ブラックボックスのような類似度関数から算出された類似度のみを利用して，データ間の類似度の階層関係に基づく木構造索引である．該類似度木は，高次元データの内部にある各次元の値を問わずに，データをまるごと利用するため,「次元の呪い」に影響されず高次元に耐えられる木構造である．また，類似度木を用いた効率的な類似検索アルゴリズムを提案して，合成データと実データを用いて提案手法の有効性と検索の効率を検証する．

本論文は以下の通りで構成される．第 2 章で関連研究を紹介する．第 3 章では，本研究の提案手法のもととなる基本原理およびその発想について詳しく説明する．第 4 章では，提案した類似度木に関わる索引の構築と木構造における操作を詳細に紹介する． 最後に，簡単なまとめと，今後の課題を述べる．

2. 関 連 研 究

類似検索に関する技術はこれまでによく研究され，主に空間索引法と，近似シグネチャー法と，局所性鋭敏型ハッシュ法 (LSH) という大まかなカテゴリに分けられる．次に，それぞれの関連手法についてまとめて紹介する．

空間索引法 について，過去三十年に渡って，多数の索引技術が提案された．早期の研究者たちは，Quadtree [8] と k-d-tree [9] をはじめ，それぞれの変種を含めて，空間における座標軸を分割する観点で多くの索引技術を提案した．一方，空間における幾何性質によりデータを分割する視点では，R-tree [6] と，その改良版 R^+-tree [10]，R^*-tree [11]，およびその他の変種を含む空間索引が提案された．そこで，代表となる 2 つ k-d-tree と R-tree について，次に簡単に説明する．

k-d-tree は，ユークリッド空間にある点のデータセットを座標軸に沿って分類する空間分割の木データ構造である．k 次元の空間において，一回毎に 1 つの座標軸に沿って，該当次元のデータ値によって，空間を二分割する．分割を一回行った結果，分割点にあるデータは空間を二分割する．再帰的に異なるデータを選びながら，空間は再帰的に分割され，最終的に二分木となる k-d-tree が構築できる．

R-tree は，最小包含矩形 (MBR) を利用して，MBR に囲まれるデータを分割しながら，さらに階層的に下位の MBRs を上位 MBR を利用して再帰的に分割することにより，最終的にお互いに重なり合う MBR で構成される木データ構造である．

上記のような空間索引手法を用いることで，従来の線形走査で行われる類似検索は，木走査に変わって，検索の効率は理想的に log オーダーとなる．しかし，次元の増加に伴って，空間索引は内部的に分割されたノード間にオーバーラップが発生するため，検索の効率は従来の線形走査より低下すると指摘された [7]．即ち，いわゆる「次元の呪い」という現象である．この理由で，既存する様々な空間索引手法は，本研究が着眼する高次元データにふさわしくない．しかも，既存の空間索引は，本論文で中心に議論するバイト列のような高次元データを扱うことができない．

近似シグネチャー法 は，高次元類似検索に関わる「次元の呪い」を解消するために提唱された索引技術である．代表的に VA-file [12] と，その変種となる IQ-tree [13] や，Landmark file [14], C^2VA-file [15], CVA-file [16] などが挙げられる．次に VA-file を代表の 1 つとして，簡単に紹介する．

VA-file は，ベクトル近似ファイル (*Vector Approximation File*) の略称であり，高次元のベクトルデータに対して，近似シグネチャーを索引としてあらかじめ作成しておくものである．VA-file を用いた類似検索は，フィルタリング (filtering) とリファインメント (refinement) との 2 段階で行う．フィルタリングでは，近似シグネチャーを利用して，距離を実計算せずに解になりそうな候補集合を取り出す．そして，リファインメントでは，候補集合から実距離を計算し，正解を絞り込む．計算量は線形的な効率であるが，単なるのフルスキャンより数倍以上速くなるため，高次元データに対する類似検索には，既存の空間索引技術よりふさわしいと 10 年間に渡って検証された [17].

VA-file 族の手法は，フルスキャンでの類似検索より速いので，空間索引手法より「次元の呪い」を回避することがある

程度できると考えられる．しかしながら，近似シグネチャーを作成するには，高次元データの内部にある各次元の値を利用する必要であるため，本論文で中心に議論するバイト列のような高次元データを扱うことができない．

局所性鋭敏型ハッシュ法 は，近似類似検索を対象として提唱されたハッシュ関数と確率的な処理に基づく近似手法である．高次元類似検索は「次元の呪い」問題に巻き込まれるため，高価なコストをかけて正確な解を求めるより，高速に近似な解を求めたほうが実用性が高いと指摘され，Arya と Mount は近似類似検索を提唱した [18]．その後，近似類似検索の解を高速に求めため，局所性鋭敏型ハッシュ法 (*Locality-Sensitive Hashing*, LSH) は提案された [19], [20]．

LSH は，ハッシュ関数族を用いて，類似したデータを高確率で同じバケットに入るようにマッピングさせる索引手法である．検索時には，同じハッシュ関数族を基にクエリデータを同様にマッピングさせたうえ，ヒットしたバケット内に入るデータを対象として，類似検索の近似解を求める．ハッシュ関数を利用することで従来の索引技術より，検索時にアクセスされるバケットおよびデータの数は大幅に削減できるが，元のデータ空間に対して適切なハッシュ関数族を設計することが非常に困難であると指摘されている [21]．これに加えて，ハッシュ関数を設計するには，高次元データの各次元を見ることが必要であり，結局「次元の呪い」に影響されることになる．この際は，類似検索の効率が低下しつつ，近似結果の精度も下がってくる．この理由で，Tao らは，検索効率を保ちながら，より精度の高い近似結果を求める LSB-tree 手法を提案した [22]．

上記に様々な既存技術が提案されたにも関わらず，本研究が着目しているバイト列のような内部構造の不明な高次元データを扱うことができない，さらにブラックボックスのように内部演算の不明な類似度関数を扱うことができないため，本研究では，新たなデータ索引手法（**類似度木**）を提案する．類似度木という索引は，ブラックボックスのような類似度関数から算出された類似度のみを利用して，データ間の類似度の階層関係に基づく木構造索引である．該類似度木は，高次元データの内部にある各次元の値を問わずに，データをまるごと利用するため,「次元の呪い」に影響されず高次元に耐えられる木構造である．次に，本研究の提案手法について詳細に述べる．

3. 提 案 手 法

本章では，類似度木を提案するにあたる要件定義および基本原理について説明する．まず，本稿で利用される記号とその意味について，表 1 に記載する．以降に特別な説明がなければ，表 1 に記載されるものに従う．

3.1 要 件 定 義

第 1 章に提起した類似検索問題 $SS(q, \delta, DB)$ に対して，下記の要件を着目して新たな提案を行う．

表 1 記 号 説 明

記号	意味
v, v_i	d 次元の数値ベクトル，$(x_1, x_2, \cdots, x_d)$
o, o_i	シリアライズ化されたデータのバイト列
f	高次元データ，または特徴量データ
f_i	i 番目の特徴量 f
δ	類似度の閾値
F	特徴量データの集合，$F = \cup_i \{f_i\}$
f_r	データ集合 F の代表
M	木構造のノード容量
$sim(\cdot, \cdot)$	データ間の類似度を算出する関数

[**要件** 1]（高次元データの内部構造が不明） 高次元データは通常に式 1 のような数値ベクトルで表すことが多いが，実応用上に処理速度を向上させるなどの観点からは，高次元データを式 2 のようなバイト列にシリアライズすることも多くある．このため，高次元データの次元数や，各次元の値を含めて内部構造が不明となる． □

$$v = (x_1, x_2, \cdots, x_d); \tag{1}$$

$$\text{byte[ ] } o = v.Serialize(); \tag{2}$$

[**要件** 2]（類似度計算関数の内部演算が不明） 類似検索を行うには，データ間の類似度 (または距離) を計算する必要である．通常には，式 1 のような数値ベクトルをデータとして，与えられた類似度関数 (一例: 式 3) を用いて，類似度の計算を行うが，実際の応用開発では，第三方から提供された開発キットを利用することが多い．この際に，式 4 のように API を呼び出して，類似度値のみを得ることになり，API の内部に埋め込んだ類似度関数の具体演算は不明となる． □

$$sim(v_1, v_2) = cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|}; \tag{3}$$

$$sim(o_1, o_2) = \text{call sim\_api}(o_1, o_2); \tag{4}$$

上述の要件は，提案をするための前提を厳しくする設定のではなく，これまでの既存研究で前提となっていた明確な数値ベクトルや，明示された距離空間，明確な類似度計算関数などの条件を緩めることである．**高次元データの内部構造が不明であっても，類似度計算関数の内部演算が不明であっても，類似検索が可能になるような緩い，かつ，応用上の制約が少なくより実践的な要件である．**

現在，類似検索システムを開発するにあたって，研究者と開発者は上記のような要件を直面することが多くあるが，類似検索向けの既存手法は適応できないため，フルスキャンを行わざるを得ない．本研究では，検索時の類似度計算量を大幅に削減するため，前述したより緩い前提向けの新たな索引手法，類似度木を提案する．本提案手法では，索引を作成するたびに，利用可能な情報が通常より少ないため，即ち，要件 1 と 2 が緩いからこそ，本提案の汎用性と実用性が高いと

考えられる．続いて，このような簡単な要件をどう取り組むかについて，類似度木を設計する基本原理を説明する．

3.2 基本原理

［**原理** 1］ データ A，B，C に対して，A と B が類似する，かつ，B と C が類似するならば，A と C が類似すると判断するという仮説が本研究の提案手法の基本原理である． □

上記の基本原理は必ずしも真ではないが，真となる可能性が高いことは，[23] に開示する観測結果で示される．真となる可能性の高い実例は挙げられる．例えば，実世界では，人物 A，B，C という 3 人がいると想定すると，A さんと B さんが友人であり，B さんと C さんが友人であるならば，A さんと C さんとも友人である可能性が高い，いわゆる Small-World 理論である [24], [25]．本研究の提案手法は，この基本原理を利用して，データ間の類似関係と，代表・被代表関係とを階層的に結び付くことを基本アイディアにする．まず，次に代表と類似について定義を述べる．

［**定義** 1］（代表） 指定された類似度閾値 δ に対して，データ f_r とデータの集合 F が，条件 $\forall f \in F, \mathrm{sim}(f_r, f) \geqq \delta$ を満たせば，f_r は F の代表という． □

定義 1 により，f_r が F の代表であることを簡単に $\mathrm{sim}(f_r, F) \geqq \delta$ を用いて表記する．

［**定義** 2］（類似） 指定された類似度閾値 δ に対して，与えられたデータ f_a と f_b との類似度が，条件 $\mathrm{sim}(f_a, f_b) \geqq \delta$ を満たせば，f_a と f_b は類似するという． □

同様に，定義 1 と 2 を合わせて，$\mathrm{sim}(f_r, F) \geqq \delta$ が成立すれば，f_r と F は類似するという．この際に，データ f_r と集合 F にあるすべてのデータと類似するという意味である．

続いて，提案手法の基本アイディアを詳細に説明する．本研究で対象として扱う高次元の特徴量データを記号 f を用いて表す．説明するために，まず次の記号を導入する．データ A，B，C の特徴量をそれぞれ f_a，f_b，f_c で表す．さらに，特徴量データの集合を F で表す．検索対象となる特徴量データを f_q で表す．f と F の関係については，下記の式で表現する．また，f_b は集合 F に属し，f_b は集合 F にあるすべての特徴量データの代表とすると想定する．そして，次の数式を用いて，上記の基本原理を表現する．

$$sim(f_a, f_b) \geqq \delta \wedge sim(f_b, f_c) \geqq \delta \Rightarrow sim(f_a, f_c) \geqq \delta \quad (5)$$

$$sim(f_q, f_b) \geqq \delta \wedge sim(f_b, f_c) \geqq \delta \Rightarrow sim(f_q, f_c) \geqq \delta \quad (6)$$

$$sim(f_q, f_b) \geqq \delta \wedge sim(f_b, F) \geqq \delta \Rightarrow sim(f_q, F) \geqq \delta \quad (7)$$

式 5 では，データ A と B が類似する，かつ，データ B と C が類似するならば，データ A と C は類似すると言える．そこに，原理 1 によって，必ずしも真だと言えないが，高い可能性で言えると仮定することである．正確には，「近似類似する」と表記すべきであるが，本論文では説明の一貫性を保つために，省略して類似すると転用する．よって，本研究で言及される類似検索の結果は，近似結果である．

式 5 に f_a を f_q に置き換えると，式 6 が成立する．そこには，クエリ q がデータ B と類似する，かつ，クエリ q がデータ C と類似するならば，クエリ q はデータ C と類似することを表す．

さらに，式 6 に f_c をデータ集合 F に置き換えると，式 7 が成立する．即ち，クエリ q がデータ B と類似する，かつ，データ B がデータ集合 F の代表であるならば，直ちにクエリ q は集合 F にあるすべてのデータと類似することを推定できる．式 7 によれば，代表関係を用いることで，クエリ q と集合 F 内のデータとの類似度を計算せずに，類似検索を近似に行うことが可能となり，類似度の計算量も大幅に削減することができると考えられる．

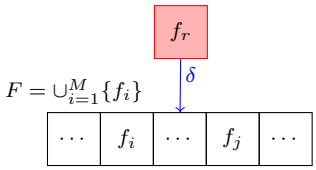

図 1 索引の基本構造

我々は，前述の原理 1 に基づいて，図 1 に示される索引の基本構造を採用して，図 2 を例とする類似度木を提案する．図 1 は，索引構造において，代表データ f_r と代表されるデータ集合 F との階層関係を表している．そこに，f_r は，エントリとして上位階層のノードに置く．F に含まれる全データをそれぞれエントリとして，F は下位階層のノードとする．f_r から F を指すリンクを用いて，f_r が F の代表であることを表す．リンクに付与される閾値 δ を用いて，定義 1 で示した条件 $\mathrm{sim}(f_r, F) \geqq \delta$ を表す．図 2 に示す索引構造は，本研究で提案する類似度木であり，図 1 の基本構造を再帰的にして，代表とデータ集合間の類似度関係を階層的に構築する木構造索引である．次の章に，図 2 を参照して類似度木の内部構造について詳しく説明する．

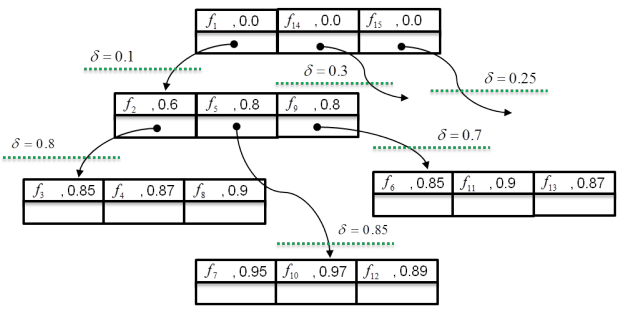

図 2 類似度木の例

4. 類似度木

本章では，類似度木の内部構造と，類似度木を構築するために必要な制約，および類似度木におけるデータの挿入，溢れ調整の操作について説明する．

4.1 内部構造

類似度木は，図 2 に示すように類似度の階層関係に基づく木構造索引である．類似度木の内部要素は，ルートノードを含む中間ノードと，最下位の階層にある葉ノードと，中間ノードのエントリと，葉ノードのエントリと，階層間の親子関係のつながりとから構成される．本稿では，基本的にディスク常駐 (disk-resident) の索引を中心に設計するが，メモリ常駐 (memory-redisent または in-memory) における実装も可能とする．

ノードはエントリの集合であるが，中間ノードと葉ノードとのエントリの内部構成は異なる．ルートノードを含む中間ノードの各エントリは，特徴量データ f_i と，f_i と上位階層の親ノードにある代表データ f_r との類似度と，f_i が代表する下位階層の子ノードを指すポインターとから構成されている．f_i が代表する下位階層ノードにデータが存在しない場合は，f_i のポインターが NULL となる．葉ノードの各エントリは，記憶媒体 (主メモリとディスクを含む) に保存されるデータを指すポインターと，指されるデータと上位階層の親ノードにある代表データ f_r との類似度とから構成される．具体的に，エントリとノードの構造を図 3 に示す擬似コードを用いて記述する．

```
Entry {
  optional Data   pRepresentative;
  required Double dSimilarity;
  required Node*  pChild = NULL;
}
Node {
  required Integer iCapacity = M;
  required Double  dThreshold;
  required Entry[] pEntries;
}
```

図 3　エントリとノードの擬似コード

ルートノードは，木構造の最上位階層にあるエントリで構成されるノードのことである．中間ノードは，ルートノードを含む，最下位階層以外のノードという．中間ノードは，木構造の最上位階層を含む中間階層にあるエントリで構成されるノードのことである．葉ノードは，木構造の最下位階層にあるエントリで構成されるノードのことである．

上位階層のノードと下位階層のノードのつながりは，類似度の階層関係にしたがう．図 2 に示すように，(上位階層) 親ノードのエントリにある特徴量データは，(下位階層) 子ノードにあるすべてのエントリの特徴量データの代表である．代表となる特徴量データが保存されているエントリと下位階層の子ノードとの間は，類似度閾値 (δ) 付きのリンクでつなぐ．代表となる特徴量データは，その下位階層の子ノードにあるすべてエントリの特徴量データとの類似度が，必ずリンクに付いている閾値 (δ) より以上でなければならない．

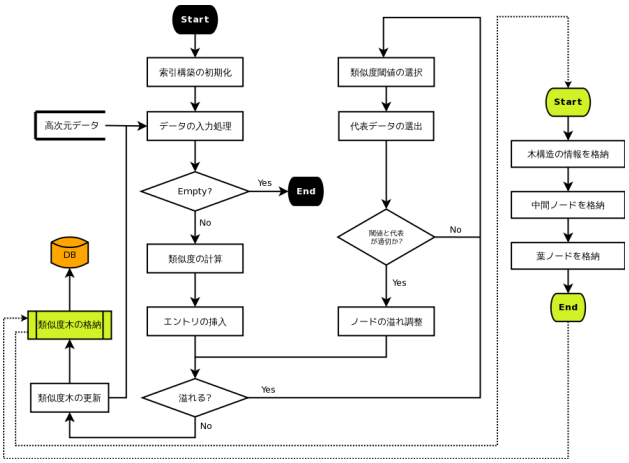

図 4　類似度木を構築するフローチャート

4.2 構造の制約

図 2 に示す類似度木の構築を保証するため，木構造に下記の制約を与える必要である．

[**制約** 1]（ノード容量 M） ノードを構成するエントリの数をノード容量という．M で表す．ノード容量が M を超える場合，ノードが溢れるという．ノードが溢れる際に，エントリをスプリットすることが発生する．エントリをスプリットするのを溢れ調整という．

[**制約** 2]（基本構造） 類似度木は必ず図 1 に示される構造を基本単位として構成される．上位ノードにあるエントリと下位ノード間のリンクに必ず類似度閾値 δ を付与する．上下位のデータ間の関係は，必ず定義 1 で示される代表関係の条件を満たす．

[**制約** 3]（閾値の階層関係） 類似度木の内部に付与される類似度閾値は，ノードの階層関係によって，大小関係が決定される．つながりのあるノードに対して，上位ノードに関わる類似度閾値は下位ノードに関わる類似度閾値より小さい．すなわち，木における類似度閾値は，上位から下位へ大きくなるような階層関係を持つ．

4.3 木の構築

類似度木の構築は，特徴量データの挿入操作に伴い，ルートノードから葉ノードまでに辿り着いて，この過程では，動的に構築するために，データの挿入や，ノード溢れ調整，代表の選出，閾値の選定などの中間処理が発生しうる．木構造の構築に関する処理の流れは，図 4 を参照して詳細に説明する．

図 4 を参照すると，類似度木の索引構築は，(黒背景白字)

Start から始まり，End までの各種処理から構成されている．そのなか，類似度木の格納という処理は，さらに（緑背景黒字）Start から始まり，End までの処理で構成されている．

これらの構成部分はそれぞれ概略次のように動作する．

索引構築の初期化という処理は，類似度木を構築する前提条件を設定するパラメーターの初期化を行う．この初期化は，木構造のノードの容量と，特徴量データの代表の選出方法と，ノードが溢れた際にノードを分割するための類似度閾値の決定方法と，木の階層の最大高さとからのパラメーターの設定を行う．

データの入力処理は，複合媒体から抽出される特徴量の高次元データを，バイト列の方式あるいは従来の数値方式で，データの挿入するたびに，１つずつ類似度木の構築流れへ渡す．

Empty という判定は，入力データのあるかを判明し，続いて入力されるデータがなければ（Empty の判定が Yes の場合），類似度木の構築処理を終了させる．入力されるデータがあれば（Empty の判定が No の場合），次の処理に引き続き行う．

類似度の計算という処理は，システムが利用可能な類似度関数もしくは類似度計算可能なＡＰＩを用いて，入力されたデータと木構造にある挿入先のノードと関連する代表データとの類似度を計算する．

エントリの挿入という処理では，前述の類似度の計算処理で算出した類似度値を用いて，前述した類似度木の構築原理にしたがって，入力されたデータを葉ノードのエントリへ挿入する．

エントリの挿入が発生した後に，ノードの容量をチェックして，溢れるかどうかを判断する．No の場合は，類似度木の更新処理を行い，適宜なタイミングに類似度木の格納という処理を介して，ディスクにあるデータベースへ索引構造とデータを書き出す．または，次のデータを挿入するように処理の流れを繰り返して，類似度木の構築を動的に行う．Yes の場合は，ノードが溢れるため，ノードを分割するような溢れ調整を行う．

溢れ調整の処理は，類似度閾値の選択と，代表データの選出と，閾値と代表が適切であるかの判断と，ノードの溢れ調整とから構成されている．溢れ調整には，適切な閾値と代表を決めるまでに，数回に繰り返して選択と選出を行う．類似度閾値の選択方法および代表データの選出方法について，具体的な実施例として節 4.5 と節 4.4 にて詳細に説明する．

類似度木の索引をディスクへ書き出す際に，内部的な処理は，木構造の情報を格納する処理部と，中間ノードを格納する処理部と，葉ノードを格納する処理部とから構成されている．処理は順次に行うことである．

4.4 代表データの選出

特徴量の代表データを選出する方法について，代表となる特徴量データの数によって，下記のような方法を実施することができる．

- 単代表 (Single Representative) と，
- 双代表 (Double Representatives) と，
- 三代表 (Triple Representatives) と，
- ・・・
- K 代表 (K Representatives) と

から代表の選出することが可能である．次に，具体的に単代表と双代表の選出の方法について説明する．

a） 単代表の選出法１：

特徴量データの集合 F において，自分とその以外のデータとの類似度の値の分散が最少となるものを代表として選出する．下記の数式を用いて表す．

$$\arg\min_{f_i \in F} \{\sigma_{f_i} | \sigma \text{が } sim(f_i, f_j) \text{ の分散}, f_j \in F\} \tag{8}$$

上記の数式に基づいて，集合 F に属する各特徴量データ f_i に対して，f_i と f_i 以外のデータとの類似度を全部算出して，類似度分布の分散値 σ が最少となるときの f_i を，集合 F の代表として選出する．

b） 単代表の選出法２：

特徴量データの集合 F において，自分が最近傍となるデータの数が最多となるものを代表として選出する．下記の数式を用いて表す．

$$\arg\max_{f_i \in F} \{RNN(f_i)\} \tag{9}$$

上記の数式に基づいて，集合 F に属する各特徴量データ f_i に対して，f_i の逆最近傍「ＲＮＮ（f_i）」を求めるうえで，逆最近傍の解の数が最も多い場合の f_i を集合 F の代表として選出する．

c） 単代表の選出法３：

特徴量データの集合 F において，特徴量データをランダム的に選択する仕方で集合 F の代表を選出する．

d） 双代表の選出法１：

特徴量データの集合 F において，類似度の値が最大となるペアの特徴量データを双代表として同時に選出する．下記の数式を用いて表す．

$$\arg\max_{(f_i, f_j)} \{sim(f_i, f_j) | f_i, f_j \in F\} \tag{10}$$

上記の数式に基づいて，集合 F に属するすべての特徴量データのペア（f_i，f_j）に対して，両データ間の類似度を算出して，類似度の値が最大となるペアを集合 F の代表として選出する．意味上は，最も固まる２つの特徴量データは代表となるような選出の仕方である．

e） 双代表の選出法２：

特徴量データの集合 F において，類似度の値が最小となるペアの特徴量データを双代表として同時に選出する．下記

の数式を用いて表す．

$$\arg \min_{(f_i,f_j)} \{sim(f_i,f_j)|f_i,f_j \in F\} \quad (11)$$

上記の数式に基づいて，集合 F に属するすべての特徴量データのペア（f_i，f_j）に対して，両データ間の類似度を算出して，類似度の値が最小となるペアを集合 F の代表として選出する．意味上は，最も離れる２つの特徴量データは代表となるような選出の仕方である．

4.5 類似度閾値の選択

類似度閾値の決め方について，ノードを分割するための類似度閾値の階層構造を構築するには，下記のような方法を実施することができる．

均等間隔と，再帰二分割と，動的な中央値とから類似度閾値の選択することが可能である．

均等間隔の選択法は，0 から 1 までの類似度範囲を等間隔に分割して，閾値を決定する方法である．例えば，0.1, 0.2, 0.3, $\cdots$, 0.9, 1.0 のように，0.1 の等間隔で $[0, 1]$ の類似度の範囲区間を 10 等分割し，類似度閾値はそれぞれの分割点の数値を取る．

再帰２分割の選択法は，0 から 1 までの類似度範囲を再帰的に log スケールに分割して，閾値を決定する方法である．例えば，１番目の閾値を 0.2 に設定して，0.2 から 1.0 までの範囲を二等分割する．そして，分割点は 0.6 となるため，２番目の閾値を 0.6 に設定する．このように二等分割を繰り返すと，0.8, 0.9, 0.95, 0.975, 0.9875, $\cdots$ などは３番目から継続の閾値系列となる．

動的な中央値の選択法は，溢れるノードにおいて，集合 F に属するすべての特徴量データと F の代表データとの類似度をそれぞれ比較して，類似度系列の中央値を閾値として決定する方法である．例えば，集合 F にある５つの特徴量データと F の代表との類似度はそれぞれ，0.5, 0.3, 0.6, 0.9, 0.95 であるとすれば，この数列の中央値は 0.6 であるため，0.6 を類似度の閾値として選出し，ノードの分割を行う．

4.6 索引の効果

上述の類似度木索引によれば，下記の効果を示す．

- 類似度木は，特徴量の形式に依存せず類似度の階層関係に基づく木構造索引を利用するため，従来の数値形式を含むバイト列形式の高次元特量をデータとして扱うことができる．
- 類似度木は，類似度の計算関数に依存せず類似度の階層関係に基づく木構造索引を利用するため，類似度関数の内部計算に依存することなく，任意の類似度関数，かつ関数の内部計算を知らずに索引の構築と類似検索を行うことができる．
- 類似度木は，類似度の階層関係に基づく木構造索引は動的に構築・更新・検索をサポートするため，データに対して動的に索引を構築・更新し，類似検索は大規模なデータに対してスケールアウトできる．
- 類似度木は，類似度の階層関係に基づく木構造索引は，索引を構築する際に高次元データの幾何空間性質に頼らずに，データの類似度の階層関係を利用するため，索引構造は特徴量データの次元数に影響されずに，索引の効果が高次元に耐えることができる．
- 類似度木は，類似度の階層関係に基づく木構造索引は，データの幾何空間性質に依存せず,「次元の呪い」の影響を受けず，索引内部にノード間のオーバーラップが発生しないことを保証するため，特徴量データの次元数の増加に伴って，検索効率を低下せず，検索の計算量を $O(\log N)$ のコストで安定的に維持することができる．

5. 実験と評価

本稿では，合成データと実データを用いて，提案した類似度木に対して，実験評価を行う．現時点で，実験は実施中であるため，最終的な評価結果は発表スライドに載せ，論文の最終版にまとめる予定である．

6. ま と め

本稿では，データの内部属性に依存せず，類似度関数または距離関数の内部計算に依存せずに，データ間の類似度の階層関係に基づいて，高次元にたえられる新たな木構造索引を提案する．

文　献

[1] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002.

[2] Kristen Grauman and Trevor Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR (1)*, pages 220–227, 2004.

[3] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Ze Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2), 2008.

[4] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *FODO*, pages 69–84, 1993.

[5] J. T. Robinson. The k-d-b-tree:a serach structure for large multidimensional dynamic indexs. In *SIGMOD*, pages 10–18, 1981.

[6] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[7] S. Berchtold, C. Böhm, D. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *ACM PODS Symposium on Principles of Database Systems*, pages 78–86, 1997.

[8] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[9] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[10] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional

objects. In *VLDB*, pages 507–518, 1987.

[11] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *SIGMOD*, pages 322–331. ACM Press, 1990.

[12] R. Weber, H. J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[13] Stefan Berchtold, Christian Böhm, H. V. Jagadish, Hans-Peter Kriegel, and Jörg Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *ICDE*, pages 577–588, 2000.

[14] Christian Böhm, Bernhard Braunmüller, Hans-Peter Kriegel, and Matthias Schubert. Efficient similarity search in digital libraries. In *Proceedings of IEEE Advances in Digital Libraries (ADL)*, pages 193–206, 2000.

[15] Hanxiong Chen, Jiyuan An, Kazutaka Furuse, and Nobuo Ohbo. C^2VA: Trim high dimensional indexes. In *WAIM*, pages 303–315, 2002.

[16] Jiyuan An, Hanxiong Chen, Kazutaka Furuse, and Nobuo Ohbo. Cva file: an index structure for high-dimensional datasets. *Knowl. Inf. Syst.*, 7(3):337–357, 2005.

[17] Stephen Blott and Roger Weber. What's wrong with high-dimensional similarity search? *PVLDB*, 1(1):3, 2008.

[18] Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, pages 271–280, 1993.

[19] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[20] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006.

[21] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD Conference*, pages 541–552, 2012.

[22] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD Conference*, pages 563–576, 2009.

[23] Wei Dong, Moses Charikar, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, pages 577–586, 2011.

[24] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[25] J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):425–443, 1969.